Task 2

Build Basic CNNs

Alexander Garcia

April 2025

Contents

1 Introduction

The goal of this phase is to build a series of basic CNNs (not using data augmentation, regularization, etc..) that performs best against validation data. Several models were brought forward from task 1 based on their accuracy with some minor adjustments and additional testing. For this task, the data was split into train, validation, and test. The layers used are for a basic CNN including Conv2D, MaxPool, GlobalAveragePool, and some Dense layers. Overall, I studied the effect of using Flatten versus GlobalAveragePooling, increasing image size, filters, layers, and implementing model checkpointing and early stopping.

2 Network Architectures

Most of the chosen network architectures were run in task 1 while attempting to overfit on the entire dataset at once without splitting. This time the models were run with training and validation to find the best one. I found the sweet spot in parameters is greater than 1 million with the models performing best between 1.6m and 2.8m. Any less or more (more without batch norm, regularization, augmentation, etc..) resulted in less accuracy.

2.1 GlobalAverage vs Flatten

I experimented with this architecture by switching Flatten for GlobalAveragePooling to check the differences.

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_25 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_27 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_26 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_28 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_27 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_29 (Conv2D)	(None, 15, 15, 256)	295,168
max_pooling2d_28 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_30 (Conv2D)	(None, 5, 5, 512)	1,180,160
max_pooling2d_29 (MaxPooling2D)	(None, 2, 2, 512)	0
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 512)	0
dense_10 (Dense)	(None, 200)	102,600
dense_11 (Dense)	(None, 8)	1,608

Total params: 1,672,784 (6.38 MB)
Trainable params: 1,672,784 (6.38 MB)
Non-trainable params: 0 (0.00 B)

(a) GlobalAveragePooling

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_4 (Conv2D)	(None, 5, 5, 512)	1,180,160
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 200)	409,800
dense_1 (Dense)	(None, 8)	1,608

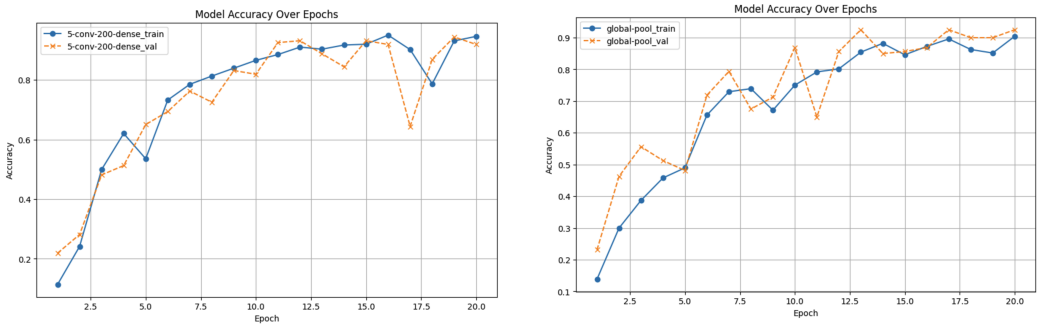
Total params: 1,979,984 (7.55 MB)
Trainable params: 1,979,984 (7.55 MB)
Non-trainable params: 0 (0.00 B)

(b) Flatten

Figure 1: Parameter count of Flatten and GlobalAveragePooling

Take note of the parameters specifically in the Dense layer. Why did this happen? Flatten works by taking the 512 output depth feature map of 2x2 and flattening all of it to 1D tensor. So to calculate, 2 x 2 x 512 = 2048 which is then passed into Dense. The calculation for that is 2048 x 200 (neurons in Dense) = 409,600 then add the bias for each dense neuron giving us the 409,800 parameters. GlobalAveragePooling has a slightly different approach where it calculates the average of each 2 x 2 feature map. Since this average is over 512 feature maps, this results in an output of (512,). To finish the calculation, I take 512 x 200 + 200 (each neuron's bias) = 102,600. However,

even with this difference in parameters, the accuracy was not greatly effected in this particular example as shown in the plots below. Keep in mind though that the larger feature maps can result in a greater difference in parameters which could then have an effect. Also note the spikes have been normal throughout training in phase 1 and 2 but the overall accuracy is the focus here. The spikes will hopefully be addressed in later phases when adding more complex methods and architectures.



(a) Flatten (b) GlobalAveragePooling

Figure 2: Accuracy comparison of Flatten and GlobalAveragePooling

2.2 Image size 150 vs 256

Another test was the image target_size parameter using the image generator. I ran two similar architectures with 5 convolution layers as seen in the previous section but using image sizes of 150x150 and 256x256. Take note of the difference in feature map sizes. Its examples like this that would result in a greater difference of parameters when using Flatten vs GlobalAveragePooling unless we adjusted the architecture to use more downsampling in the larger image size example.

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_25 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_27 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_26 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_28 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_27 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_29 (Conv2D)	(None, 15, 15, 256)	295,168
max_pooling2d_28 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_30 (Conv2D)	(None, 5, 5, 512)	1,180,160
max_pooling2d_29 (MaxPooling2D)	(None, 2, 2, 512)	0
global_average_pooling2d_4 (GlobalAveragePooling2D)	(None, 512)	0
dense_10 (Dense)	(None, 200)	102,600
dense_11 (Dense)	(None, 8)	1,608

Total params: 1,672,784 (6.38 MB)
Trainable params: 1,672,784 (6.38 MB)
Non-trainable params: 0 (0.00 B)

(a) 150x150

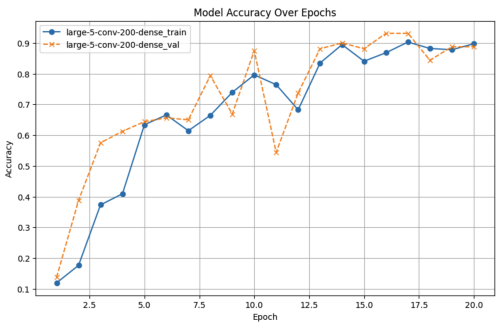
Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_15 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_17 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_16 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_18 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_17 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_19 (Conv2D)	(None, 28, 28, 256)	295,168
max_pooling2d_18 (MaxPooling2D)	(None, 14, 14, 256)	0
conv2d_20 (Conv2D)	(None, 12, 12, 512)	1,180,160
max_pooling2d_19 (MaxPooling2D)	(None, 6, 6, 512)	0
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 512)	0
dense_6 (Dense)	(None, 200)	102,600
dense_7 (Dense)	(None, 8)	1,608

Total params: 1,672,784 (6.38 MB)
Trainable params: 1,672,784 (6.38 MB)
Non-trainable params: 0 (0.00 B)

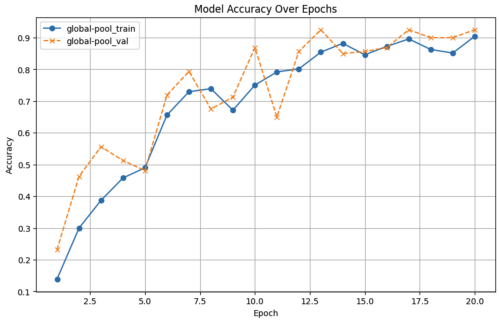
(b) 256x256

Figure 3: Comparing 150 vs 256 starting image size

Looking at the charts below there was no considerable change in accuracy when testing the larger image size relative to the original 150. However, the curve is quite a bit smoother on the lower image size which may be a result of the lack of smooth downsampling in the final MaxPooling layer for the larger image. The GlobalPooling is selecting from a 3x larger feature map before going into the dense layer.



(a) 256 starting image size

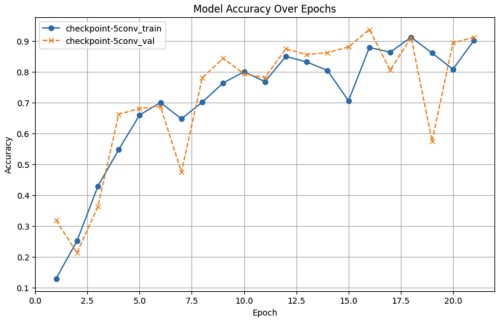


(b) 150 starting image size

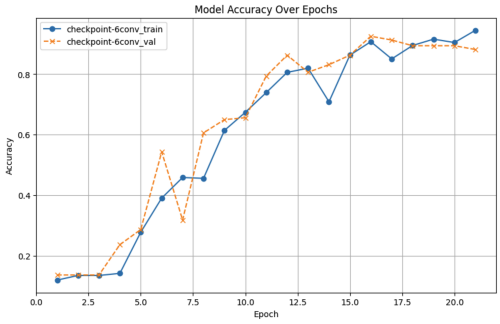
Figure 4: Accuracy comparison of starting image sizes

2.3 Model Checkpoint - Early Stop

After running several models and checking the validation accuracy I took the best two and implemented callbacks. The monitor metric for both early-stopping and model checkpoint is validation loss. I also set the patience parameter to 5 epochs in early stopping because my results were spiking. I did not want the model to stop on one weird spike in validation loss because I felt the spikes were useful information for me to learn. I also implemented the early stop to restore the best weights because of this. Meaning, if the model continued to run on patience and deteriorated or had a spike, I would still have the best performing weights at the end of a previous epoch. The two models I ran these callbacks on are a 5 and 6 convolutional layer model. Below is a side-by-side comparison of the model accuracies.



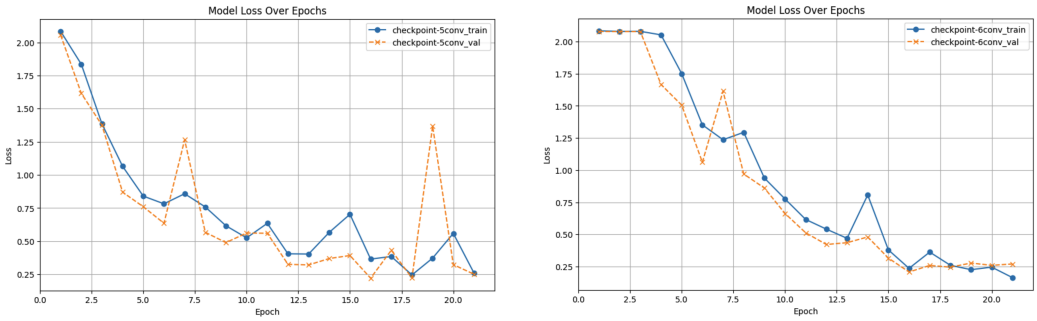
(a) 5 convolution accuracy graph



(b) 6 convolution accuracy graph

Figure 5: Accuracy comparison of potential best models

The interesting thing that continues to arise on this dataset while using these simpler models is the spikes. They occur in both training and validation, which leads me to believe that a pattern the model is learning breaks when finding certain images. My guess for the culprits are the images of a freshly updated town hall while the rest of the village is still of the previous level. Nonetheless, the accuracy on the test set is sufficient enough for now to proceed to the next steps. A similar pattern is observed in the loss graphs below



(a) 5 convolution loss graph

(b) 6 convolution loss graph

Figure 6: Loss comparison of potential best models

3 Evaluating Best Model On Test

After implementing the early-stopping and model-checkpointing, the best weights came from the 6 convolution layer with 1 dense layer having 1000 neurons bringing the total parameters of the model to 2.6 million. The amount of parameters in the dense layer is 513k which is only 19% of the total parameters, meaning the convolutions are still accounting for a majority of the work as desired. For a refresh the architecture is pictured below

Layer (type)	Output Shape	Param #
conv2d_140 (Conv2D)	(None, 148, 148, 32)	896
conv2d_141 (Conv2D)	(None, 146, 146, 64)	18,496
max_pooling2d_93 (MaxPooling2D)	(None, 73, 73, 64)	0
conv2d_142 (Conv2D)	(None, 71, 71, 128)	73,856
max_pooling2d_94 (MaxPooling2D)	(None, 35, 35, 128)	0
conv2d_143 (Conv2D)	(None, 33, 33, 256)	295,168
max_pooling2d_95 (MaxPooling2D)	(None, 16, 16, 256)	0
conv2d_144 (Conv2D)	(None, 14, 14, 256)	590,080
max_pooling2d_96 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_145 (Conv2D)	(None, 5, 5, 512)	1,180,160
max_pooling2d_97 (MaxPooling2D)	(None, 2, 2, 512)	0
global_average_pooling2d_15 (GlobalAveragePooling2D)	(None, 512)	0
dense_55 (Dense)	(None, 1000)	513,000
dense_56 (Dense)	(None, 8)	8,008

Total params: 2,679,664 (10.22 MB)
Trainable params: 2,679,664 (10.22 MB)
Non-trainable params: 0 (0.00 B)

Figure 7: Best model

Keep in mind this model was run with the patience and restore best weights parameters so even though it had spikes and eventually broke accuracy I captured its best weight and therefore those were saved in the model.keras file. When evaluating on the test set the model had an accuracy of 92.5% and loss of 0.20. All things considered I thought this was great. Below is a table and graph of each classes performance so I could understand the impact of each class more on their own.

Interestingly enough, the theory of the model learning patterns that might break when seeing images of a base, like Town Hall 17 containing lots of

Table 1: Test Performance

Class	Precision	Recall	F1-Score
th_10	0.9524	1.0000	0.9756
th_11	0.9524	1.0000	0.9756
th_12	1.0000	1.0000	1.0000
th_13	0.9524	1.0000	0.9756
th_14	1.0000	0.8500	0.9189
th_15	1.0000	1.0000	1.0000
th_16	1.0000	0.5000	0.6667
th_17	0.7000	1.0000	0.8235
Macro Avg	0.9446	0.9188	0.9170

Figure 8: Metrics Per Class

Town Hall 16 objects, seems to be confirmed when looking at the metrics. Specifically, look at recall and precision metrics of Town Hall 16 and Town Hall 17. Town Hall 17 is relatively new, released Nov 25 2024, so the building upgrades are not all available yet. Meaning, as of writing this, there are still some structures which have not received an update so they are still Town Hall 16 level and cannot be upgraded until the creator Super Cell releases a new update. It seems possible some Town Hall 16s are being classified as 17 due to their similar looks and having more Town Hall 17 samples than 16.

Table 2: Image Counts per Town Hall Level

Town Hall Level	Number of Images
Town Hall 10	130
Town Hall 11	130
Town Hall 12	130
Town Hall 13	130
Town Hall 14	130
Town Hall 15	142
Town Hall 16	117
Town Hall 17	133
Total	**1,042**

4 Conclusion

In summary, I have identified my best model to use in the next phases of augmentation, regularization and addition of other tools. Overall the model is performing well on test data despite having fluctuations in the training and validation phases. I took note of the differences in Flatten compared to GlobalAveragePooling layers and its effect on parameters as well as different image sizes and model architectures. Testing these did not change my choice final model as it was still run with GlobalAveragePooling and 150 x 150 image size but noting the differences it could have will do well for future projects and learning. It appears at this point the slight imbalance of Town Hall 16 may be having an adverse effect which will hopefully be corrected in the following phases and tasks.